

Tcl manual page - Tcl Built-In Commands

 tcl.tk/man/tcl/TclCmd/Tcl.htm

NAME

Tcl — Tool Command Language

SYNOPSIS

Summary of Tcl language syntax.

DESCRIPTION

The following rules define the syntax and semantics of the Tcl language:

[1] Commands.

A Tcl script is a string containing one or more commands. Semi-colons and newlines are command separators unless quoted as described below. Close brackets are command terminators during command substitution (see below) unless quoted.

[2] Evaluation.

A command is evaluated in two steps. First, the Tcl interpreter breaks the command into *words* and performs substitutions as described below. These substitutions are performed in the same way for all commands. Secondly, the first word is used to locate a command procedure to carry out the command, then all of the words of the command are passed to the command procedure. The command procedure is free to interpret each of its words in any way it likes, such as an integer, variable name, list, or Tcl script. Different commands interpret their words differently.

[3] Words.

Words of a command are separated by white space (except for newlines, which are command separators).

[4] Double quotes.

If the first character of a word is double-quote ("") then the word is terminated by the next double-quote character. If semi-colons, close brackets, or white space characters (including newlines) appear between the quotes then they are treated as ordinary characters and included in the word. Command substitution, variable substitution, and backslash substitution are performed on the characters between the quotes as described below. The double-quotes are not retained as part of the word.

[5] Argument expansion.

If a word starts with the string {"*} followed by a non-whitespace character, then the leading {"*} is removed and the rest of the word is parsed and substituted as any other word. After substitution, the word is parsed as a list (without command or variable substitutions; backslash substitutions are performed as is normal for a list

and individual internal words may be surrounded by either braces or double-quote characters), and its words are added to the command being substituted. For instance, “cmd a {*} {b [c]} d {*} {\$e f {g h}}” is equivalent to “cmd a b {[c]} d {\$e} f {g h}”.

[6] Braces.

If the first character of a word is an open brace (“{”) and rule [5] does not apply, then the word is terminated by the matching close brace (“}`). Braces nest within the word: for each additional open brace there must be an additional close brace (however, if an open brace or close brace within the word is quoted with a backslash then it is not counted in locating the matching close brace). No substitutions are performed on the characters between the braces except for backslash-newline substitutions described below, nor do semi-colons, newlines, close brackets, or white space receive any special interpretation. The word will consist of exactly the characters between the outer braces, not including the braces themselves.

[7] Command substitution.

If a word contains an open bracket (“[”) then Tcl performs *command substitution*. To do this it invokes the Tcl interpreter recursively to process the characters following the open bracket as a Tcl script. The script may contain any number of commands and must be terminated by a close bracket (“]”). The result of the script (i.e. the result of its last command) is substituted into the word in place of the brackets and all of the characters between them. There may be any number of command substitutions in a single word. Command substitution is not performed on words enclosed in braces.

[8] Variable substitution.

If a word contains a dollar-sign (“\$”) followed by one of the forms described below, then Tcl performs *variable substitution*: the dollar-sign and the following characters are replaced in the word by the value of a variable. Variable substitution may take any of the following forms:

\$name

Name is the name of a scalar variable; the name is a sequence of one or more characters that are a letter, digit, underscore, or namespace separators (two or more colons). Letters and digits are *only* the standard ASCII ones (**0–9**, **A–Z** and **a–z**).

\$name(index)

Name gives the name of an array variable and *index* gives the name of an element within that array. *Name* must contain only letters, digits, underscores, and namespace separators, and may be an empty string. Letters and digits are *only* the standard ASCII ones (**0–9**, **A–Z** and **a–z**). Command substitutions, variable substitutions, and backslash substitutions are performed on the characters of *index*.

\${name}

Name is the name of a scalar variable or array element. It may contain any characters whatsoever except for close braces. It indicates an array element if

name is in the form “*arrayName(index)*” where *arrayName* does not contain any open parenthesis characters, “(”, or close brace characters, “}”, and *index* can be any sequence of characters except for close brace characters. No further substitutions are performed during the parsing of *name*.

There may be any number of variable substitutions in a single word. Variable substitution is not performed on words enclosed in braces.

Note that variables may contain character sequences other than those listed above, but in that case other mechanisms must be used to access them (e.g., via the **set** command's single-argument form).

[9] Backslash substitution.

If a backslash (“\”) appears within a word then *backslash substitution* occurs. In all cases but those described below the backslash is dropped and the following character is treated as an ordinary character and included in the word. This allows characters such as double quotes, close brackets, and dollar signs to be included in words without triggering special processing. The following table lists the backslash sequences that are handled specially, along with the value that replaces each sequence.

\a

Audible alert (bell) (Unicode U+000007).

\b

Backspace (Unicode U+000008).

\f

Form feed (Unicode U+00000C).

\n

Newline (Unicode U+00000A).

\r

Carriage-return (Unicode U+00000D).

\t

Tab (Unicode U+000009).

\v

Vertical tab (Unicode U+00000B).

\<newline>whiteSpace

A single space character replaces the backslash, newline, and all spaces and tabs after the newline. This backslash sequence is unique in that it is replaced in a separate preprocess before the command is actually parsed. This means that it will be replaced even when it occurs between braces, and the resulting space will be treated as a word separator if it is not in braces or quotes.

\|

Backslash (“\”).

\ooo

The digits *ooo* (one, two, or three of them) give a eight-bit octal value for the Unicode character that will be inserted, in the range 000–377 (i.e., the range U+000000–U+0000FF). The parser will stop just before this range overflows, or when the maximum of three digits is reached. The upper bits of the Unicode character will be 0.

\xhh

The hexadecimal digits *hh* (one or two of them) give an eight-bit hexadecimal value for the Unicode character that will be inserted. The upper bits of the Unicode character will be 0 (i.e., the character will be in the range U+000000–U+0000FF).

\uhhhh

The hexadecimal digits *hhhh* (one, two, three, or four of them) give a sixteen-bit hexadecimal value for the Unicode character that will be inserted. The upper bits of the Unicode character will be 0 (i.e., the character will be in the range U+000000–U+00FFFF).

\Uhhhhhhhh

The hexadecimal digits *hhhhhhhh* (one up to eight of them) give a twenty-one-bit hexadecimal value for the Unicode character that will be inserted, in the range U+000000–U+10FFFF. The parser will stop just before this range overflows, or when the maximum of eight digits is reached. The upper bits of the Unicode character will be 0.

The range U+010000–U+10FFFD is reserved for the future.

Backslash substitution is not performed on words enclosed in braces, except for backslash-newline as described above.

[10] Comments.

If a hash character (“#”) appears at a point where Tcl is expecting the first character of the first word of a command, then the hash character and the characters that follow it, up through the next newline, are treated as a comment and ignored. The comment character only has significance when it appears at the beginning of a command.

[11] Order of substitution.

Each character is processed exactly once by the Tcl interpreter as part of creating the words of a command. For example, if variable substitution occurs then no further substitutions are performed on the value of the variable; the value is inserted into the word verbatim. If command substitution occurs then the nested command is processed entirely by the recursive call to the Tcl interpreter; no substitutions are performed before making the recursive call and no additional substitutions are performed on the result of the nested script.

Substitutions take place from left to right, and each substitution is evaluated completely before attempting to evaluate the next. Thus, a sequence like

```
set y [set x 0][incr x][incr x]
```

will always set the variable *y* to the value, 012.

[12] Substitution and word boundaries.

Substitutions do not affect the word boundaries of a command, except for argument expansion as specified in rule [5]. For example, during variable substitution the entire value of the variable becomes part of a single word, even if the variable's value contains spaces.